USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

APELLIDOS: **NOMBRE :**

Desarrollo de Aplicaciones Web

21 de mayo de 2025

Puntuación:

La puntuación total es de 6 puntos, distribuidos según se indica en cada pregunta.

---

i) ¿Quién fue el creador del HTML y en qué contexto? ¿Qué 2 tipos de etiquetas lo componen y cuáles son sus características a nivel de renderizado? ¿Cómo indicamos que un documento está en HTML 5? ¿Mediante qué instrucción se indica la codificación de un documento? (0,2 ptos).

ii) ¿Cuáles son los dos componentes básicos de una regla de estilos CSS y qué papel desempeñan? ¿Cuál es el uso de la etiqueta *@import* en un documento CSS? ¿Para qué se utiliza la propiedad "*box-sizing*" en un documento CSS? ¿Cuáles serían sus posibles valores y cuál sería su efecto? Indica las diferencias y similitudes entre los atributos id, class y name de una etiqueta HTML. (0,6 ptos).

iii) Dado el código HTML de la Fig. 3.1, indica en la tabla adjunta, mediante el uso de las letras N (negro) y R (rojo), ¿cuál sería el color de los diferentes partes del texto una vez renderizado por un navegador? (0,4 ptos) sometido únicamente a la regla que aparece cada columna de la tabla adjunta.

{color:red};	div > p	div + p	h2 + p	h2 ~ p
USC				
ETSE				
DAW				
3er Curso				
Santiago				

iv) Dado el código adjunto y sin modificarlo:

```
<body>
    <em id="p1"></em>
    <em id="p2"></em>
</body>
```

Crea las reglas CSS necesarias para que utilizando "float" y "relative" el resultado sean los dos cuadrados con dimensiones 100 x 100 aparecidos en las Fig. 4.1 y Fig. 4.2. Las reglas CSS deberán dividirse en dos ficheros externos llamados *cssBasico.css* y *cssPos.css*. En *cssBásico.css* se incluirán las reglas básicas relativas a formato de los cuadrados, incluyendo su tamaño y color. En *cssPos.css* se incluirán exclusivamente las reglas relativas al posicionamiento de los cuadrados. Solamente *cssBasico.css* será llamado desde el fichero html (0,8 ptos).

Nota → *El cuadrado azul hace referencia a p1, el cuadrado rojo hace referencia a p2.*

v) El código HTML siguiente:

```
<body>
      <h2>USC. ETSE.</h2>
      <p></p>
      <p>DAW</p>
</body>
```

junto con un fichero de estilos generan la Figura 5. Existe un código *javascript* que mediante el uso de tres funciones (*miFuncion_1(), miFuncion_2()* y *miFuncion_3()*) permite dotar a la página de la siguiente funcionalidad:

miFuncion_1() → Al poner el ratón sobre el cuadrado, cambia el texto "DAW" a "Compostela" y viceversa, dependiendo de cual aparezca escrito.

miFunción_2() → Al hacer doble *click* sobre el texto ("DAW" o "Compostela") modifica el formato del cuadrado (reduce el tamaño a 50 x 50, pasa el color a rojo y el borde lo deja punteado), siguiendo las reglas de la clase "*miClase*" de la hoja de estilos que tiene enlazado. El proceso es reversible, de tal manera que al hacer doble *click* de nuevo se restaura el formato original y viceversa.

miFunción_3() → Al hacer *click* sobre el texto ("USC. ETSE." modifica el color del cuadrado, siguiendo el siguiente esquema:

i) Si el cuadrado es 100 x 100, el color alternará entre azul y verde

ii) Si el cuadrado es 50 x 50, el color alternará entre rojo y verde

Escribir el código CSS de la regla para ***miClase*** y el código ***javascript*** necesario para poder obtener la funcionalidad descrita. (0,8 ptos)

vi) Dada la siguiente expresión regular (0,2 ptos):

[5-9](?=[l-r]+)

indica cuál es el resultado de su captura sobre el siguiente texto:

"El código buscado es: f648r4q1m2h5"

vii) Dado el siguiente código HTML

```
<h2>Escuela Técnica de Ingeniería</h2>
<ul>
  <li>Ingeniería Quimica</li>
  <li>Ingeniería Ambiental</li>
  <li>Ingeniería Informática</li>
</ul>
```

Escribe el código jQuery necesario para que:

a) El texto "***Ingeniería Ambiental***" aparezca en azul (por defecto).
b) Al poner el ratón encima de "Escuela Técnica de Ingeniería", el texto "***Ingeniería Informática***" se ponga en verde
c) Al hacer *click* en el texto "***Ingeniería Química***", el texto de cabecera pase de ser "Escuela Técnica de Ingeniería" a ser "***Facultad de Física***". (0,6 ptos).

viii) Indica a qué corresponde el código que aparece en las Figuras 8.1 y 8.2. Explica línea a línea dicho código. Dado el fichero *json* de la figura 8.3, escribe el código necesario para que el renderizado del código *html* adjunto, resulte en la Figura 8.4 (0,3 ptos).

```
<body>
     <h3>Escuela Técnica Superior de Ingeniería</h3>
     <ul id="datos"></ul>
</body>
```

ix) ¿Qué se entiende por Aplicación Web? (0,1 ptos).

x) ¿Qué se entiende por JSP y a qué lenguaje de programación está ligado? ¿Qué se entiende por "Lenguaje de Expresión y qué es necesario hacer para poder utilizarlo en un fichero JSP? ¿Qué se entiende por JSTL y que es necesario hacer para poder utilizarlo en una aplicación web? (0,3 ptos).

xi) La Figura 11.1 representa el esquema de una aplicación web que corre sobre máquinas situadas en diferentes emplazamientos. Dicha aplicación está pensada para dar de alta usuarios junto a sus preferencias. Para dicho proceso, se crearán dos listas por separado que se irán actualizando con cada alta de usuario. La primera lista (*lista local*) consiste en un *ArrayList* en donde se introducirán los usuarios dados de alta en cada localización. La segunda lista (*lista global*) consiste en un *HashMap* en donde se introducirán todos los usuarios dados de alta en los diferentes emplazamientos, siendo accesibles mediante el ID de cada usuario que hará de clave.

Cada usuario deberá darse de alta con un ID, que se supone único. La particularidad de este sistema es que, durante el proceso de darse de alta, no se verificará si el usuario está dado ya de alta en la *lista local*, por lo que en dicha lista podrá haber usuarios repetidos. Sin embargo, en la *lista global*, solo se incluirán usuarios que tengan IDs diferentes. Por tanto, no existirán usuarios repetidos.

Para el desarrollo de la aplicación se han utilizado 4 servlets, 7 vistas (incluyendo la página de inicio), 1 ficheros jsp adicional y el fichero de inicio *daw.jsp*, con las siguientes características:

SERVLETS

a) ***servlet_1*** *(Registro.java)* → Hace de controlador en la parte de "registro" de un usuario. Para ello:

i) Crea un usuario apoyándose en el método ***getUsuario()***. Este método realiza las siguientes acciones:

   a. Instancia un objeto de la clase Usuario
   b. Da valor a las propiedades nombre, apellido, Id y password del objeto, con los datos procedentes del formulario *formInput* (ver Fig. 11.5).

ii) Guarda el usuario actual en el atributo de request "currentUser".
iii) Crea y envía la cookie *userID,* donde guarda el Id del usuario actual.
iv) Invoca a los servlets *servlet_3* y *servlet_4*.
v) Pasa el control a la vista *welcomeUser.jsp* (ver Fig. 11.6)

b) ***servlet_2*** *(Preferencias.java)* → Realiza las siguientes acciones:

i) Recoge, mediante el método *getExtras()* las preferencias seleccionadas por un usuario dado de alta a través del formulario *formPreferencias* (ver Fig 11.7).
ii) Inserta los extras en el usuario actual
iii) Pasa el control a una vista (*vistaExtras.jsp*) (ver Fig 11.8).

c) ***servlet_3*** *(RegistroUsuarios.java)* → Procesa la *lista local* de usuarios para mantenerla actualizada en todo momento. Crea las acciones necesarias para dar soporte a la vista *vistaUsuarios* (ver Fig. 11.9)

d) ***servlet_4*** *(RegistroGeneral.java)* → Procesa la lista global de usuarios para mantenerla actualizada en todo momento. Crea las acciones necesarias para dar soporte a la vista *vistaDatos* (ver Fig 11.10).

VISTAS

Todas las vistas están construidas mediante lenguaje de expresión (EL) y etiquetas JSTL. Además, utilizan acciones JSP para su construcción, lo cual permite usar la misma cabecera/pie en todas ellas, mediante la inclusión de los ficheros: *header.jsp* y *footer.html*. A mayores, algunas de las vistas incluyen formularios provenientes de diferentes ficheros html. En concreto:

i) **daw.jsp** (Fig. 11.2) → Menú inicial desplegable (Fig. 11.3) proporcionado por el fichero *formInput.html*. Permite seleccionar una entre las diferentes acciones proporcionadas. Si se selecciona la opción "Cambiar de preferencias", devuelve la página inicial con un aviso en rojo (Fig. 11.4).

ii) ***registro.jsp*** (Fig. 11.5) → Conjunto de cajas de texto, proporcionadas por el fichero *formRegistro.html*, utilizadas para dar de alta al usuario.

iii) ***welcomeUser.jsp*** (Fig. 11.6) → Página de bienvenida al usuario, indicándole cuántos usuarios están dados de alta localmente en ese momento. Abre la opción de añadir extras para el usuario actual, pulsando "*Pulsa aquí*".

iv) ***preferencias.jsp*** (Fig. 11.7) → Página que incluye una lista de opciones para el usuario al darse de alta. Dicha lista es proporcionada por el fichero *formPreferencias.html*.

v) **vistaExtras.jsp** (Fig. 11.8) → Página que informa al usuario de las opciones que ha elegido, una vez procesadas.

vi) ***vistaUsuarios*** (Fig. 11.9) → Página que informa del nombre y apellidos de los usuarios dados de alta localmente.

vii) **vistaDatos** (Fig. 11.10) → Página que informa sobre el número total de usuarios registrados globalmente. Además, incluye la información de cada uno de los usuarios.

Fichero welcome.jsp → Fichero jsp cuyo papel es gestionar las diferentes acciones seleccionadas por el usuario a partir del formulario de entrada.

Para construir la aplicación, se han almacenado los ficheros en las siguientes carpetas:

- Carpeta / → *daw.jsp, welcome.jsp, welcomeUser.jsp, preferencias.jsp.*
- Carpeta /estilos → *básico.css*
- Carpeta /includes/otros → *header.html, footer.html*
- Carpeta /includes/formularios → *formInput.html, formPreferencias.html y formRegistro.html*
- Carpeta /vistas → *registro.jsp, vistaDatos.jsp, vistaExtras.jsp y vistaUsuarios.jsp*

Además, todas las clases de java estarán dentro de una subcarpeta llamada daw, en el lugar correspondiente.

Se pide implementar:

i) Código de *welcome.jsp*

ii) Código del *servlet_1*

iii) Códgio del *servlet_3*

iv) Código de *vistaUsuarios.jsp*

v) Código del fichero de despliegue web.xml.

vi) Código del método *setUsuario()* de la clase *ListadoGeneral*. Dicho método debe introducir usuarios no repetidos en una lista creada mediante un *HashMap*, en donde la clave será el *ID* del usuario y el valor el objeto *Usuario* correspondiente.

Nota 1.- Las invocaciones a las vistas realizadas desde los *servlets* se realizarán necesariamente utilizando objetos de la clase *Despachador* (ver anexo).

Nota 2.- Los diagramas de las diferentes clases que intervienen, así como el código HTML de los formularios figuran en el anexo proporcionado.

El valor de la pregunta es 2 ptos.